Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №8

Технології паралельних обчислень

Тема: Розробка алгоритмів для
розподілених систем клієнт-серверної архітектури

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірила:
Стеценко І.В.

Київ 2024

ЗМІСТ

1 ЗАВДАННЯ

1. Розробити веб-застосування клієнт-серверної архітектури, що реалізує алгоритм множення матриць або інший, який був Вами реалізований в рамках курсу «Технології паралельних обчислень», на стороні сервера з використанням паралельних обчислень. Розгляньте два варіанти реалізації 1) дані для обчислень знаходяться на сервері та 2) дані для обчислень знаходяться на клієнтській частині застосування. 60 балів.
2. Дослідити швидкість виконання запиту користувача при різних обсягах даних. 30 балів.
3. Порівняти реалізацію алгоритму в клієнт-серверній системі та в розподіленій системі з рівноправними процесорами. 10 балів.

2 ВИКОНАННЯ

2.1 Структура проєкту

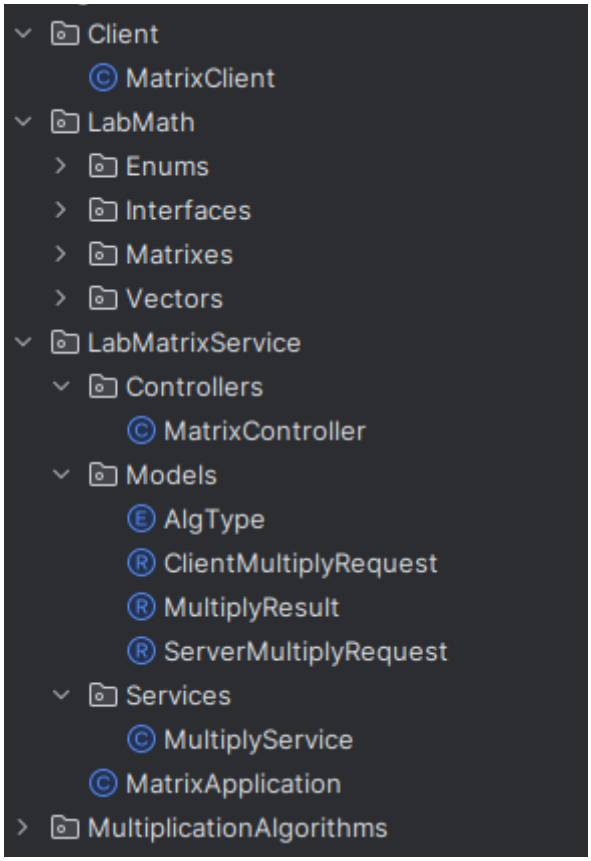


Рисунок 2.1.1 - Структура проєкту

Загально проєкт складається з двох частин: Client та Server.

Загалом сервер складається з математичної бібліотеки та алгоритмів, які були описані в минулих комп’ютерних практикумах, тому сконцентруємося на головному.

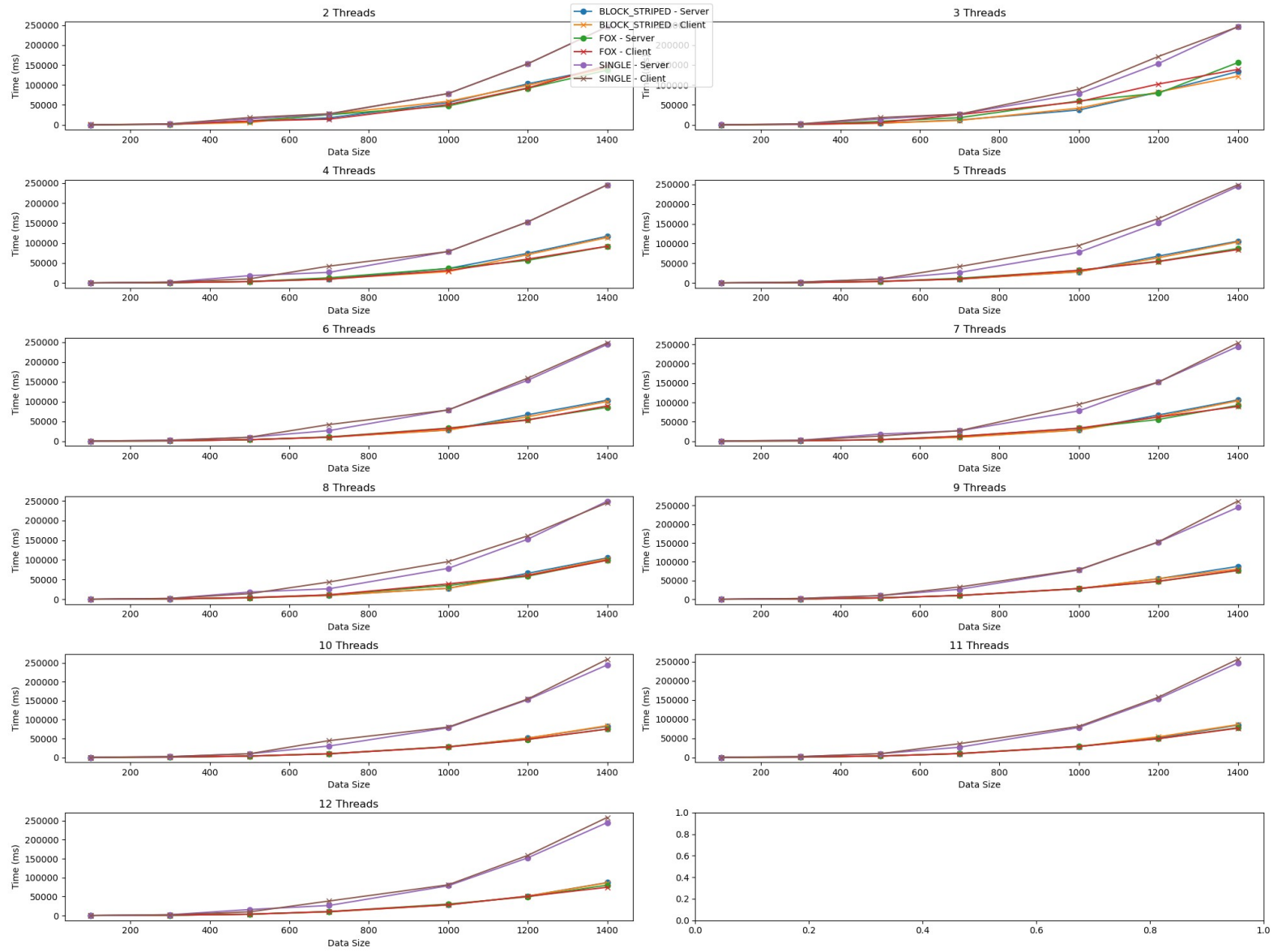
Поглянемо на LabMatrixService.

Controllers — API-контролери, що реагують на запити та викликають сервіси для користувача.

Models — модуль, що включає в себе перелік типів аргументів, класи запитів, обгортки над класом матриці тощо.

Services — модуль, що включає сервіс множення матриць.

2.2 Результати



Наданий графік демонструє продуктивність трьох різних алгоритмів множення матриць—BLOCK STRIPED, FOX та SINGLE—за різних розмірів даних і кількості потоків. Продуктивність вимірюється у часі (мілісекундах) і розділена на два типи обчислень: Server, де генерація матриць і множення відбуваються на сервері, і Client, де матриці відсилаються на сервер для множення.

Спостереження:

1. **Масштабованість зі збільшенням кількості потоків:** Усі алгоритми показують покращення продуктивності зі збільшенням

кількості потоків, до певного моменту. Це покращення більш виражене для обчислень на стороні сервера, ніж для обчислень на стороні клієнта. Зменшення приросту продуктивності за певної кількості потоків свідчить про те, що алгоритми, ймовірно, досягають ліміту через такі фактори, як накладні витрати на управління потоками або обмеження апаратного забезпечення.**Різниця в продуктивності алгоритмів:**

- Алгоритм BLOCK STRIPED здається добре масштабується зі збільшенням кількості потоків, що вказує на його можливу вигоду від паралельного обчислення.
- Алгоритм FOX також показує покращення з більшою кількістю потоків, але не так суттєво, як BLOCK STRIPED, що може свідчити про вищі накладні витрати або менш оптимальний розподіл роботи між потоками.
- Алгоритм SINGLE, як очікувалося, показує найменше покращення з додатковими потоками.

2. **Обчислення на стороні сервера проти сторони клієнта:**
   - Для всіх алгоритмів і кількості потоків обчислення на стороні сервера послідовно швидші, ніж на стороні клієнта. Ця різниця підкреслює накладні витрати, пов'язані з передачею матриць на сервер для множення. Час, витрачений на передачу даних через мережу, ймовірно, додає значну затримку порівняно з виконанням множення безпосередньо на сервері, де дані генеруються.

3 ВИСНОВОК

Ефективність множення матриць у розподіленому обчислювальному середовищі значно впливає на вибір алгоритму, кількість потоків та місце виконання обчислень. Оптимізовані алгоритми, як-от BLOCK STRIPED, які розроблені для паралельної обробки, можуть ефективно використовувати кілька потоків для покращення продуктивності. Однак, накладні витрати у клієнт-серверних моделях можуть погіршити продуктивність, що підкреслює важливість мінімізації передачі даних або оптимізації стратегій обробки даних для розподілених систем. Для досягнення оптимальної продуктивності важливо враховувати ці фактори при проєктуванні та розгортанні завдань розподіленого обчислення.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-113 курсу

Панченка С. В

```
// ../Server/src/main/java/org/LabMatrixService/MatrixApplication.java

package org.LabMatrixService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MatrixApplication {
    public static void main(String[] args) {
        SpringApplication.run(MatrixApplication.class, args);
    }
}



// ../Server/src/main/java/org/LabMatrixService/Models/AlgType.java

package org.LabMatrixService.Models;

public enum AlgType {
        BLOCK_STRIPED,
        FOX,
    SINGLE
}



// ../Server/src/main/java/org/LabMatrixService/Models/MultiplyResult.java

package org.LabMatrixService.Models;

import org.LabMath.Matrixes.Matrix2D;
```

```
public record MultiplyResult(Matrix2D result) {}


// ../Server/src/main/java/org/LabMatrixService/Models/ClientMultiplyRequest.java

package org.LabMatrixService.Models;

import org.LabMath.Matrixes.Matrix2D;

public record ClientMultiplyRequest(AlgType algType, Matrix2D first, Matrix2D
second, int threadsNum) {
}


// ../Server/src/main/java/org/LabMatrixService/Models/ServerMultiplyRequest.java

package org.LabMatrixService.Models;

public record ServerMultiplyRequest(AlgType algType, int threadsNum, int size) {
}


// ../Server/src/main/java/org/LabMatrixService/Controllers/MatrixController.java

package org.LabMatrixService.Controllers;

import org.LabMath.Matrixes.Matrix2DFactory;
import org.LabMatrixService.Models.ClientMultiplyRequest;
import org.LabMatrixService.Services.MultiplyService;
import org.LabMatrixService.Models.ServerMultiplyRequest;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/api")
public class MatrixController {

    @PostMapping("/hello")
    public ResponseEntity<?> hello() {
        return ResponseEntity.ok("Hello World!");
    }

    @PostMapping("/multiply_server")
     public ResponseEntity<?> multiplyServer(@RequestBody ServerMultiplyRequest
request) {
        var matrixFactory = new Matrix2DFactory();

        var minVal = 0;
        var maxVal = 10;

            var first = matrixFactory.getRandom(request.size(), request.size(), minVal,
maxVal);
          var second = matrixFactory.getRandom(request.size(), request.size(), minVal,
maxVal);

                    return ResponseEntity.ok(MultiplyService.solve(request.algType(),
request.threadsNum(), first, second));
    }

    @PostMapping("/multiply_client")
     public ResponseEntity<?> multiplyClient(@RequestBody ClientMultiplyRequest
```

```
request) {
                    return    ResponseEntity.ok(MultiplyService.solve(request.algType(),
request.threadsNum(), request.first(), request.second()));
    }
}


// ../Server/src/main/java/org/LabMatrixService/Services/MultiplyService.java

package org.LabMatrixService.Services;


import org.LabMath.Matrixes.Matrix2D;
import org.LabMatrixService.Models.AlgType;
import org.MultiplicationAlgorithms.BlockStriped.BlockStripedAlgorithm;
import org.MultiplicationAlgorithms.Fox.FoxAlgorithm;

public class MultiplyService {
    public MultiplyService() {}

     public static Matrix2D solve(AlgType algType, int threadsNum, Matrix2D first,
Matrix2D second) {
        return switch(algType) {
                                                case    BLOCK_STRIPED    ->    new
BlockStripedAlgorithm().multiply(threadsNum, first, second);
            case FOX -> new FoxAlgorithm().multiply(threadsNum, first, second);
            case SINGLE -> first.getMul(second);
        };
    }
}
```

```
// ../Server/src/main/java/org/MultiplicationAlgorithms/MultiplyAlgo.java

package org.MultiplicationAlgorithms;

import org.LabMath.Matrixes.Matrix2D;

public interface MultiplyAlgo {
	Matrix2D multiply(int threadsNum, Matrix2D first, Matrix2D second);
}


// ../Server/src/main/java/org/MultiplicationAlgorithms/Fox/FoxAlgorithm.java

package org.MultiplicationAlgorithms.Fox;

import org.LabMath.Matrixes.Matrix2D;
import org.MultiplicationAlgorithms.MultiplyAlgo;

import java.util.ArrayList;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class FoxAlgorithm implements MultiplyAlgo {
	public FoxAlgorithm() {}

	@Override
	public Matrix2D multiply(int countThread, Matrix2D first, Matrix2D second) {
		var complexSize = (int) Math.sqrt(countThread - 1) + 1;
		var complexFirst = MatrixToComplexMatrix(first, complexSize);
		var complexSecond = MatrixToComplexMatrix(second, complexSize);

		var innerSize = complexFirst[0][0].getCols();
```

```
        var complex = new Matrix2D[complexSize][complexSize];
        for(var i = 0; i < complexSize; ++i) {
            for(var j = 0; j < complexSize; ++j) {
                complex[i][j] = new Matrix2D(innerSize, innerSize);
            }
        }

        var executor = Executors.newFixedThreadPool(countThread);
        for(var k = 0; k < complexSize; ++k) {
            var futures = new ArrayList<Future<Matrix2D>>();
            for(var i = 0; i < complexSize; ++i) {
                for(var j = 0; j < complexSize; ++j) {
                    var index = (i + k) % complexSize;
                    var task = new FoxAlgorithmTask(
                            complexFirst[i][index],
                            complexSecond[index][j],
                            complex[i][j]);
                    futures.add(executor.submit(task));
                }
            }

            for(var i = 0; i < complexSize; ++i) {
                for(var j = 0; j < complexSize; ++j) {
                    try {
                        complex[i][j] = futures.get(i * complexSize +
j).get();
                    } catch (Exception ignored) {}
                }
            }
        }
        executor.shutdown();
        return ComplexMatrixToMatrix(complex, first.getRows(),
```

```
second.getCols());
    }

    private Matrix2D[][] MatrixToComplexMatrix(Matrix2D matrix, int size){
        var complexMatrix = new Matrix2D[size][size];
        var rows = matrix.getRows();
        var cols = matrix.getCols();
        var inner = ((cols - 1) / size) + 1;
        for(var i = 0; i < size; ++i) {
            for(var j = 0; j < size; ++j) {
                complexMatrix[i][j] = new Matrix2D(inner, inner);
                var local = complexMatrix[i][j];
                for(var k = 0; k < inner; ++k) {
                    for (var l = 0; l < inner; ++l) {
                        var rowIndex = i * inner + k;
                        var colIndex = j * inner + l;
                        if(rowIndex >= rows || colIndex >= cols){
                            local.setAt(0, k, l);
                            continue;
                        }
                        var element = matrix.getAt(rowIndex, colIndex);
                        local.setAt(element, k, l);
                    }
                }
            }
        }
        return complexMatrix;
    }

    private Matrix2D ComplexMatrixToMatrix(Matrix2D[][] complexMatrix, int
rows, int cols) {
        var matrix = new Matrix2D(rows, cols);
```

```
            for(var i = 0; i < complexMatrix.length; ++i) {
                for(var j = 0; j < complexMatrix[i].length; ++j) {
                    var local = complexMatrix[i][j];
                    var localRows = local.getRows();
                    var localCols = local.getCols();
                    for (var k = 0; k < localRows; ++k) {
                        for (var l = 0; l < localCols; ++l) {
                            var rowIndex = i * localRows + k;
                            var colIndex = j * localCols + l;
                            if(rowIndex >= rows || colIndex >= cols) continue;
                            var el = local.getAt(k, l);
                            matrix.setAt(el, rowIndex, colIndex);
                        }
                    }
                }
            }

            return matrix;
        }
}
```

```
// ../Server/src/main/java/org/MultiplicationAlgorithms/Fox/FoxAlgorithmTask.java

package org.MultiplicationAlgorithms.Fox;



import org.LabMath.Matrixes.Matrix2D;

import java.util.concurrent.Callable;
```

```
public class FoxAlgorithmTask implements Callable<Matrix2D> {
	private final Matrix2D first;
	private final Matrix2D second;
	private final Matrix2D result;

	public FoxAlgorithmTask(Matrix2D first, Matrix2D second, Matrix2D result){
		this.first = first;
		this.second = second;
		this.result = result;
	}

	@Override
	public Matrix2D call(){
		result.add(first.getMul(second));
		return result;
	}
}
```

```
//                    ../Server/src/main/java/org/MultiplicationAlgorithms/BlockStriped/
BlockStripedAlgorithm.java

package org.MultiplicationAlgorithms.BlockStriped;

import org.LabMath.Matrixes.Matrix2D;
import org.MultiplicationAlgorithms.MultiplyAlgo;

import java.util.ArrayList;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class BlockStripedAlgorithm implements MultiplyAlgo {
```

```
    protected static final String ERROR_MULTIPLICATION = "Rows and columns are not equal";
    protected static final String ERROR_NUM_OF_THREADS = "Number of threads must be positive";

    public BlockStripedAlgorithm() {}

    @Override
    public Matrix2D multiply(int threadsNum, Matrix2D first, Matrix2D second) {
        if(threadsNum <= 0) {
            throw new IllegalArgumentException(ERROR_NUM_OF_THREADS);
        }

        var firstRows = first.getRows();
        var firstCols = first.getCols();
        var secondRows = second.getRows();
        var secondCols = second.getCols();

        if(firstCols != secondRows) {
            throw new IllegalArgumentException(ERROR_MULTIPLICATION);
        }

        var result = new Matrix2D(firstRows, secondCols);

        var executor = Executors.newFixedThreadPool(threadsNum);
        var callables = new ArrayList<BlockStripedAlgorithmTask>();
        var futures = new ArrayList<Future<Double>>();

        for(var i = 0; i < secondCols; ++i) {
            //System.out.println("Iteration: " + i);
```

```
                for(var row = 0; row < firstRows; ++row) {
                        var col = row - i;
                        col = col < 0 ? col + secondCols : col;
                        //System.out.println("\tProc: " + row + "\tCol: " + col);
                        var task = new BlockStripedAlgorithmTask(row, col, first,
second);
                        callables.add(task);
                }
                try {
                        futures.addAll(executor.invokeAll(callables));
                        callables.clear();
                } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                }

        }
        executor.shutdown();
        try {
                for(var i = 0; i < secondCols; ++i) {
                        for(var row = 0; row < firstRows; ++row) {
                                var col = row - i;
                                col = col < 0 ? col + secondCols : col;
                                var future = futures.get(i * firstRows + row);
                                result.setAt(future.get(), row, col);
                        }
                }
        } catch (InterruptedException | ExecutionException e) {
                throw new RuntimeException(e);
        }

        return result;
    }
```

```
}

//                ../Server/src/main/java/org/MultiplicationAlgorithms/BlockStriped/
BlockStripedAlgorithmTask.java

package org.MultiplicationAlgorithms.BlockStriped;

import org.LabMath.Matrixes.Matrix2D;

import java.util.concurrent.Callable;

public class BlockStripedAlgorithmTask implements Callable<Double> {
    private final Matrix2D first;
    private final Matrix2D second;
    private final int row;
    private final int col;

    public BlockStripedAlgorithmTask(int row, int col, Matrix2D first, Matrix2D
second) {
        this.first = first;
        this.second = second;
        this.row = row;
        this.col = col;
    }

    @Override
    public Double call(){
        var element = 0.0;
        for(var i = 0; i < first.getCols(); ++i) {
            element += first.getAt(row, i) * second.getAt(i, col);
        }
        return element;
```

```
        }

}


// ../Server/src/main/java/org/LabMath/Enums/Coords.java

package org.LabMath.Enums;

public enum Coords {
    Y,
    X
}


// ../Server/src/main/java/org/LabMath/Matrixes/Matrix2D.java

package org.LabMath.Matrixes;

import jdk.jshell.spi.ExecutionControl;
import org.LabMath.Interfaces.MathMatrix;

public class Matrix2D implements MathMatrix<Matrix2D> {
    private static final String ERROR_MULTIPLICATION = "Rows and columns are
not equal";
    private static final String ERROR_INDEXES = "Indexes are less than 0";
    private final int rows;
    private final int cols;
    private final GeneralMatrix mat;

    public static void main(String[] args) {}
```

```
    @Override
    public String toString() {
        return mat.toString();
    }

    public Matrix2D(int rows, int cols) {
        mat = new GeneralMatrix(rows, cols);
        this.rows = rows;
        this.cols = cols;
    }

    public int getRows() {
        return this.rows;
    }

    public int getCols() {
        return this.cols;
    }

    @Override
    public void add(Matrix2D other) {
        this.mat.add(other.mat);
    }

    @Override
                        public void div(Matrix2D other) throws
ExecutionControl.NotImplementedException {
        throw new ExecutionControl.NotImplementedException("");
    }

    @Override
    public void add(double value) {
```

```
    this.mat.add(value);
}

@Override
public void div(double value) {
    this.mat.div(value);
}

@Override
public void mul(double value) {
    this.mat.mul(value);
}

@Override
public void sub(double value) {
    this.mat.sub(value);
}

@Override
public Matrix2D getMul(Matrix2D other) {
    var cols = getCols();

    assert cols == other.getRows() : ERROR_MULTIPLICATION;

    var result = new Matrix2D(rows, cols);

    for(var i = 0; i < getRows(); ++i) {
        for(var j = 0; j < other.getCols(); ++j) {
            var value = 0.0;
            for(var k = 0; k < cols; ++k) {
                value += this.mat.getAt(i, k) * other.mat.getAt(k, j);
            }
```

```
            result.setAt(value, i, j);
        }
    }

    return result;
}

@Override
public void set(Matrix2D other) {
    this.mat.set(other.mat);
}

@Override
public void sub(Matrix2D other) {
    this.mat.sub(other.mat);
}

@Override
public int[] getDimensions() {
    return this.mat.getDimensions();
}

@Override
public double getAt(int... indexes) {
    if(indexes.length != 2) {
        throw new IllegalArgumentException(ERROR_INDEXES);
    }
    return this.mat.getAt(indexes);
}

@Override
public void setAt(double value, int... indexes) {
```

```
        if(indexes.length != 2) {
            throw new IllegalArgumentException(ERROR_INDEXES);
        }
        this.mat.setAt(value, indexes);
    }

    @Override
    public int calcIndex(int... indexes) {
        return this.mat.calcIndex(indexes);
    }

    @Override
    public int compareTo(Matrix2D o) {
        return this.mat.compareTo(o.mat);
    }
}


// ../Server/src/main/java/org/LabMath/Matrixes/GeneralMatrix.java

package org.LabMath.Matrixes;

import jdk.jshell.spi.ExecutionControl;
import org.LabMath.Interfaces.MathMatrix;
import org.LabMath.Vectors.GeneralVector;

import java.util.Arrays;

public final class GeneralMatrix implements MathMatrix<GeneralMatrix> {
    private static final String ERROR_INDEXES = "Indexes are less than 0";
        private static final String ERROR_DIMENSIONS = "Matrix dimensions not
equal";
```

```
    private static final String ERROR_DIMENSION_INDEXES = "Indexes length is not equal to amount of dimension";
    private final int[] dimensions;
    private final int total;
    private final GeneralVector mat;

    public GeneralMatrix(int... dimensions) {
        this.dimensions = dimensions.clone();
        var t = 1;
        for(var d : dimensions) t *= d;
        this.total = t;
        this.mat = new GeneralVector(this.total);
    }

    private String doDraw(int[] indexes, int dimension) {
        var res = new StringBuilder();
        res.append("{");
        for(var i = 0; i < this.dimensions[dimension]; ++i) {
            indexes[dimension] = i;
            if(dimension == this.dimensions.length - 1) {
                res.append(this.mat.getAt(calcIndex(indexes)));
            } else {
                res.append(doDraw(indexes, dimension + 1));
            }
            res.append(this.dimensions[dimension] - 1 == i ? "" : ", ");
        }
        res.append("}");
        return res.toString();
    }

    @Override
    public String toString() {
```

```
    var indexes = new int[this.dimensions.length];
    return doDraw(indexes, 0);
}

private void checkDimensions(int[] dimensions) {
    if(!Arrays.equals(this.dimensions, dimensions)) {
        throw new IllegalArgumentException(ERROR_DIMENSIONS);
    }
}

private void checkIndexes(int[] indexes) {
    if(!Arrays.stream(indexes).allMatch(e -> e >= 0)) {
        throw new IllegalArgumentException(ERROR_INDEXES);
    }
}

@Override
public void add(GeneralMatrix other) {
    checkDimensions(other.dimensions);
    for(var i = 0; i < total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + other.mat.getAt(i));
    }
}

@Override
public void add(double value) {
    for(var i = 0; i < this.total; ++i) {
        this.mat.setAt(i, this.mat.getAt(i) + value);
    }
}

@Override
```

```
    public void div(double value) {
        for(var i = 0; i < this.total; ++i) {
            this.mat.setAt(i, this.mat.getAt(i) / value);
        }
    }

    @Override
    public void mul(double value) {
        for(var i = 0; i < this.total; ++i) {
            this.mat.setAt(i, this.mat.getAt(i) * value);
        }
    }

    @Override
    public void sub(double value) {
        for(var i = 0; i < this.total; ++i) {
            this.mat.setAt(i, this.mat.getAt(i) - value);
        }
    }

    @Override
                public GeneralMatrix getMul(GeneralMatrix other) throws
ExecutionControl.NotImplementedException {
        throw new ExecutionControl.NotImplementedException("");
    }

    @Override
    public void set(GeneralMatrix other) {
        checkDimensions(other.dimensions);
        for(var i = 0; i < this.total; ++i) {
            this.mat.setAt(i, this.mat.getAt(i));
        }
```

```
    }

    @Override
    public void sub(GeneralMatrix other) {
        checkDimensions(other.dimensions);
        for(var i = 0; i < total; ++i) {
            this.mat.setAt(i, other.mat.getAt(i) - other.mat.getAt(i));
        }
    }

    @Override
    public int[] getDimensions() {
        return dimensions.clone();
    }

    @Override
    public double getAt(int... indexes) {
        checkIndexes(indexes);
        return this.mat.getAt(this.calcIndex(indexes));
    }

    @Override
    public void setAt(double value, int... indexes) {
        checkIndexes(indexes);
        var index = this.calcIndex(indexes);
        this.mat.setAt(index, value);
    }

    @Override
                        public void div(GeneralMatrix other) throws
ExecutionControl.NotImplementedException {
        throw new ExecutionControl.NotImplementedException("");
```

```
    }

    @Override
    public int calcIndex(int... indexes) {
        if(indexes.length != dimensions.length) {
            throw new IllegalArgumentException(ERROR_DIMENSION_INDEXES);
        }
        var index = 0;
        var mult = 1;
        for(var i : dimensions) mult *= i;
        for(var i = 0; i < indexes.length; ++i) {
            mult /= dimensions[i];
            index += indexes[i] * mult;
        }
        return index;
    }

    @Override
    public int compareTo(GeneralMatrix o) {
        for(var i = 0; i < total; ++i) {
            if((Math.abs(this.mat.getAt(i) - o.mat.getAt(i)) > 0.000000001)) {
                return 1;
            }
        }
        return 0;
    }
}


// ../Server/src/main/java/org/LabMath/Matrixes/Matrix2DFactory.java

package org.LabMath.Matrixes;
```

```
public class Matrix2DFactory {

    public Matrix2DFactory() {}

    public static void main(String[] args) {
        var factory = new Matrix2DFactory();
        var minVal = 0;
        var maxVal = 10;
        var rows = 5;
        var cols = 6;
        var one = factory.getRandom(rows, cols, minVal, maxVal);
        var two = factory.getRandom(cols, rows, minVal, maxVal);
        var result = one.getMul(two);
        System.out.println(result);
    }

    public Matrix2D getRandom(int rows, int cols, int minVal, int maxVal) {
        var res = new Matrix2D(rows, cols);
        for(var i = 0; i < rows; ++i) {
            for(var j = 0; j < cols; ++j) {
                res.setAt(Math.random() * (maxVal - minVal) + minVal, i, j);
            }
        }
        return res;
    }
}


// ../Server/src/main/java/org/LabMath/Interfaces/MathMatrix.java

package org.LabMath.Interfaces;
```

```
import org.LabMath.Interfaces.General.*;

public interface MathMatrix<T> extends Cloneable, Addable<T>, Subtractable<T>,
Divisible<T>, Comparable<T>,
        GetMultipliable<T>, Settable<T>, DoubleSubtractable, DoubleMultipliable,
DoubleAddable, DoubleDivisible {
   int[] getDimensions();
   double getAt(int... indexes);
   void setAt(double value, int... indexes);
   int calcIndex(int... indexes);
}


// ../Server/src/main/java/org/LabMath/Interfaces/MathVector.java

package org.LabMath.Interfaces;

import org.LabMath.Interfaces.General.*;

public interface MathVector<T> extends Cloneable, Divisible<T>, Multipliable<T>,
Addable<T>, Subtractable<T>,
      DoubleDivisible, DoubleMultipliable, DoubleAddable, DoubleSubtractable {
   double getSize();
   double getSizeSquared();
   double getDotProduct(T other);
   double getDistance(T other);
   T getForwardVector();
   double getAt(int index);
   void setAt(int index, double value);
   T getOpposite();
   void toOpposite();
```

```
    void set(T other);
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/GetMultipliable.java
```

```
package org.LabMath.Interfaces.General;
```

```
import jdk.jshell.spi.ExecutionControl;
```

```
public interface GetMultipliable<T> {
   T getMul(T other) throws ExecutionControl.NotImplementedException;
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/DoubleMultipliable.java
```

```
package org.LabMath.Interfaces.General;
```

```
public interface DoubleMultipliable {
    void mul(double other);
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/Divisible.java
```

```
package org.LabMath.Interfaces.General;
```

```
import jdk.jshell.spi.ExecutionControl;
```

```
public interface Divisible<T> {
    void div(T other) throws ExecutionControl.NotImplementedException;
```

```
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/DoubleDivisible.java

package org.LabMath.Interfaces.General;

public interface DoubleDivisible {
    void div(double other);
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/Multipliable.java

package org.LabMath.Interfaces.General;

public interface Multipliable<T> {
    void mul(T other);
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/Subtractable.java

package org.LabMath.Interfaces.General;

public interface Subtractable<T> {
    void sub(T other);
}
```

```
// ../Server/src/main/java/org/LabMath/Interfaces/General/Addable.java
```

```
package org.LabMath.Interfaces.General;

public interface Addable<T> {
    void add(T other);
}


// ../Server/src/main/java/org/LabMath/Interfaces/General/DoubleAddable.java

package org.LabMath.Interfaces.General;

public interface DoubleAddable {
    void add(double other);
}


// ../Server/src/main/java/org/LabMath/Interfaces/General/DoubleSubtractable.java

package org.LabMath.Interfaces.General;

public interface DoubleSubtractable {
    void sub(double other);
}


// ../Server/src/main/java/org/LabMath/Interfaces/General/Settable.java

package org.LabMath.Interfaces.General;

public interface Settable<T> {
    void set(T other);
}
```

```
// ../Server/src/main/java/org/LabMath/Vectors/Vector2D.java

package org.LabMath.Vectors;

import org.LabMath.Enums.*;
import org.LabMath.Interfaces.MathVector;

public class Vector2D implements MathVector<Vector2D> {
    private final GeneralVector vec = new GeneralVector(2);

    public Vector2D() {}

    public Vector2D(double y, double x) {
        set(y, x);
    }

    public Vector2D(Vector2D other) {
        set(other);
    }

    public double getX() {
        return getAt(Coords.X.ordinal());
    }

    public double getY() {
        return getAt(Coords.Y.ordinal());
    }

    public void set(Vector2D other) {
        vec.set(other.vec);
```

```
}

public void set(double y, double x) {
    setX(x);
    setY(y);
}

public void setX(double value) {
    setAt(Coords.X.ordinal(), value);
}

public void setY(double value) {
    setAt(Coords.Y.ordinal(), value);
}

@Override
public Vector2D clone() {
    return new Vector2D(getY(), getX());
}

@Override
public String toString() {
    return vec.toString();
}

@Override
public void add(Vector2D other) {
    vec.add(other.vec);
}

@Override
public void add(double value) {
```

```
    vec.add(value);
}

@Override
public void sub(double value) {
    vec.sub(value);
}

@Override
public void sub(Vector2D other) {
    vec.sub(other.vec);
}

@Override
public void mul(Vector2D other) {
    vec.mul(other.vec);
}

@Override
public void mul(double value) {
    vec.mul(value);
}

@Override
public void div(Vector2D other) {
    vec.div(other.vec);
}

@Override
public void div(double value) {
    vec.div(value);
}
```

```
@Override
public double getSize() {
    return vec.getSize();
}

@Override
public double getSizeSquared() {
    return vec.getSizeSquared();
}

@Override
public double getDotProduct(Vector2D other) {
    return vec.getDotProduct(other.vec);
}

@Override
public double getDistance(Vector2D other) {
    return vec.getDistance(other.vec);
}

@Override
public Vector2D getForwardVector() {
    var forwardVec = clone();
    forwardVec.vec.set(forwardVec.vec.getForwardVector());
    return forwardVec;
}

@Override
public double getAt(int index) {
    return vec.getAt(index);
}
```

```
    @Override
    public void setAt(int index, double value) {
        vec.setAt(index, value);
    }

    @Override
    public Vector2D getOpposite() {
        var v = clone();
        v.toOpposite();
        return v;
    }

    @Override
    public void toOpposite() {
        vec.toOpposite();
    }
}


// ../Server/src/main/java/org/LabMath/Vectors/GeneralVector.java

package org.LabMath.Vectors;

import org.LabMath.Interfaces.MathVector;

import java.util.Arrays;

public class GeneralVector implements MathVector<GeneralVector> {
        private static final String ERROR_LENGTHS_NOT_EQUAL = "Lengths of
points arguments are not equal";
    private double[] arguments;
```

```
public GeneralVector(int length) {
    setLength(length);
}

public GeneralVector(GeneralVector other) {
    setLength(other.getLength());
    set(other);
}

public int getLength() {
    if(arguments == null) {
        return 0;
    }
    return arguments.length;
}

public void setLength(int length) {
    var currentLength = getLength();

    if(currentLength==length) return;

    var minLength = Math.min(currentLength, length);
    var args = new double[length];
    for(var i = 0; i < minLength; ++i) {
        args[i] = getAt(i);
    }

    arguments = args;
}

@Override
```

```
public void set(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, other.getAt(i));
    }
}

@Override
public GeneralVector clone() {
    return new GeneralVector(this);
}

@Override
public String toString() {
    return Arrays.toString(arguments);
}

private void checkSizesEqual(GeneralVector other) {
    assert getLength() == other.getLength() : ERROR_LENGTHS_NOT_EQUAL;
}

@Override
public void add(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) + other.getAt(i));
    }
}

@Override
public void add(double value) {
    for(var i = 0; i < getLength(); ++i) {
```

```
        setAt(i, getAt(i) + value);
    }
}

@Override
public void sub(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) - other.getAt(i));
    }
}

@Override
public void sub(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) - value);
    }
}

@Override
public void mul(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * other.getAt(i));
    }
}

@Override
public void mul(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) * value);
    }
```

```
}

@Override
public void div(GeneralVector other) {
    checkSizesEqual(other);
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / other.getAt(i));
    }
}

@Override
public void div(double value) {
    for(var i = 0; i < getLength(); ++i) {
        setAt(i, getAt(i) / value);
    }
}

@Override
public double getSize() {
    return Math.sqrt(getSizeSquared());
}

@Override
public double getSizeSquared() {
    double s = 0;
    for(var i = 0; i < getLength(); ++i) {
        s += Math.pow(getAt(i), 2);
    }
    return s;
}

@Override
```

```
public double getDotProduct(GeneralVector other) {
    checkSizesEqual(other);
    var prod = 0;
    for(var i = 0; i < getLength(); ++i) {
        prod += getAt(i) * other.getAt(i);
    }
    return prod;
}

@Override
public double getDistance(GeneralVector other) {
    checkSizesEqual(other);
    var dist = 0.0;
    for(var i = 0; i < getLength(); ++i) {
        dist += Math.pow(getAt(i) - other.getAt(i), 2);
    }
    dist = Math.sqrt(dist);
    return dist;
}

@Override
public GeneralVector getForwardVector() {
    var forwardVec = clone();
    var size = getSize();
    for(var i = 0; i < getLength(); ++i) {
        forwardVec.setAt(i, getAt(i) / size);
    }
    return forwardVec;
}

@Override
public double getAt(int index) {
```

```
        return arguments[index];
    }

    @Override
    public void setAt(int index, double value) {
        arguments[index] = value;
    }

    @Override
    public GeneralVector getOpposite() {
        var v = clone();
        v.toOpposite();
        return v;
    }

    @Override
    public void toOpposite() {
        for(var i = 0; i < getLength(); ++i) {
            setAt(i, -getAt(i));
        }
    }

}


// ../Server/src/main/java/org/Client/MatrixClient.java

package org.Client;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.LabMath.Matrixes.Matrix2D;
```

```
import org.LabMath.Matrixes.Matrix2DFactory;
import org.LabMatrixService.Models.AlgType;
import org.LabMatrixService.Models.ClientMultiplyRequest;
import org.LabMatrixService.Models.ServerMultiplyRequest;

import java.io.FileWriter;
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpRequest.BodyPublishers;
import java.net.http.HttpResponse.BodyHandlers;
import java.util.ArrayList;
import java.util.List;

public class MatrixClient {

    public static void main(String[] args) throws Exception {
        var client = HttpClient.newHttpClient();

        var threadsNums = new int[] {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
//          var matrixSizes = new int[] {100, 300, 500, 700, 1000, 1200, 1400, 1500,
1700, 1800, 2000};
        var matrixSizes = new int[] {200, 400, 600};

        // String header = "MillisServer,MillisClient,Size,ThreadsNum,AlgType\n";

        try (FileWriter fileWriter = new FileWriter("records.csv")) {
            // fileWriter.append(header);

            for(var size : matrixSizes) {
```

```
            var matCreateStat = createMatrices(size);
            for(var threadsNum : threadsNums) {
                for(var algType : AlgType.values()) {
                    var clientReqMillis = sendClientMultiply(client, algType,
                            matCreateStat.first, matCreateStat.second, threadsNum);
                                var serverReqMillis = sendServerMultiply(client, algType,
threadsNum, size);
                    fileWriter.append(String.valueOf(serverReqMillis))
                            .append(",")
                                                    .append(String.valueOf(clientReqMillis +
matCreateStat.millis()))
                            .append(",")
                            .append(String.valueOf(size))
                            .append(",")
                            .append(String.valueOf(threadsNum))
                            .append(",")
                            .append(algType.toString())
                            .append("\n");
                }
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
  }

   private record MatrixCreationStat(Matrix2D first, Matrix2D second, long millis)
{}

  static MatrixCreationStat createMatrices(int size) {
    var minVal = 1;
```

```
    var maxVal = 10;
    var matrixFactory = new Matrix2DFactory();

    var matrixCreationStart = System.currentTimeMillis();
    var first = matrixFactory.getRandom(size, size, minVal, maxVal);
    var second = matrixFactory.getRandom(size, size, minVal, maxVal);
    var matrixCreationDur = System.currentTimeMillis() - matrixCreationStart;

    return new MatrixCreationStat(first, second, matrixCreationDur);
}

static long sendClientMultiply(HttpClient client, AlgType algType,
                    Matrix2D first, Matrix2D second, int threadsNum)
        throws IOException, InterruptedException {
    var cur = System.currentTimeMillis();

    var clientReq = new ClientMultiplyRequest(
            algType, first, second, threadsNum
    );
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonRequest = objectMapper.writeValueAsString(clientReq);

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("http://localhost:8080/api/multiply_client"))
            .header("Content-Type", "application/json")
            .POST(BodyPublishers.ofString(jsonRequest))
            .build();

    client.send(request, BodyHandlers.ofString());

    return System.currentTimeMillis() - cur;
}
```

```
        static long sendServerMultiply(HttpClient client, AlgType algType, int threadsNum, int size)
        throws IOException, InterruptedException {
    var cur = System.currentTimeMillis();

    var clientReq = new ServerMultiplyRequest(algType, threadsNum, size);
    ObjectMapper objectMapper = new ObjectMapper();
    String jsonRequest = objectMapper.writeValueAsString(clientReq);

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("http://localhost:8080/api/multiply_server"))
            .header("Content-Type", "application/json")
            .POST(BodyPublishers.ofString(jsonRequest))
            .build();

    client.send(request, BodyHandlers.ofString());

    return System.currentTimeMillis() - cur;
  }
}
```